Red Hat build of MicroShift 4.15

Configuring

Configuring MicroShift

Last Updated: 2024-06-07

Red Hat build of MicroShift 4.15 Configuring

Configuring MicroShift

Legal Notice

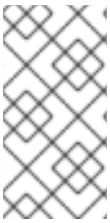
Abstract

This document provides instructions for configuring MicroShift.

Table of Contents

CHAPTER 1. HOW CONFIGURATION TOOLS WORK

A YAML file customizes MicroShift instances with your preferences, settings, and parameters.



NOTE

If you want to make configuration changes or deploy applications through the MicroShift API with tools other than **kustomize** manifests, you must wait until the Greenboot health checks have finished. This ensures that your changes are not lost if Greenboot rolls your **rpm-ostree** system back to an earlier state.

1.1. DEFAULT SETTINGS

If you do not create a **config.yaml** file, default values are used. The following example shows the default configuration settings.

- To see the default values, run the following command:

```
$ microshift show-config
```

Default values example output in YAML form

```
dns:
  baseDomain: microshift.example.com (1)
network:
  clusterNetwork:
    - 10.42.0.0/16 (2)
  serviceNetwork:
    - 10.43.0.0/16 (3)
  serviceNodePortRange: 30000-32767 (4)
node:
  hostnameOverride: "" (5)
  nodeIP: "" (6)
apiServer:
  advertiseAddress: 10.44.0.0/32 (7)
  subjectAltNames: [] (8)
debugging:
  logLevel: "Normal" (9)
```

(1) Base domain of the cluster. All managed DNS records will be subdomains of this base.

(2) A block of IP addresses from which Pod IP addresses are allocated.

(3) A block of virtual IP addresses for Kubernetes services.

(4) The port range allowed for Kubernetes services of type NodePort.

(5) The name of the node. The default value is the hostname.

(6) The IP address of the node. The default value is the IP address of the default route.

(7) A string that specifies the IP address from which the API server is advertised to members of the cluster. The default value is calculated based on the address of the service network.

8 Subject Alternative Names for API server certificates.

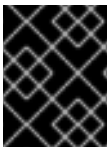
9 Log verbosity. Valid values for this field are **Normal**, **Debug**, **Trace**, or **TraceAll**.

1.2. USING A YAML CONFIGURATION FILE

On start up, MicroShift searches the system-wide **/etc/microshift/** directory for a configuration file named **config.yaml**. To use custom configurations, you must create the configuration file and specify any settings that are expected to override the defaults before starting MicroShift.

1.2.1. Custom settings

To create custom configurations, you must create a **config.yaml** file in the **/etc/microshift/** directory, and then change any settings that are expected to override the defaults before starting or restarting MicroShift.

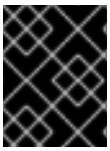


IMPORTANT

Restart MicroShift after changing any configuration settings to have them take effect. The **config.yaml** file is read only when MicroShift starts.

1.2.2. Configuring the advertise address network flag

The **apiserver.advertiseAddress** flag specifies the IP address on which to advertise the API server to members of the cluster. This address must be reachable by the cluster. You can set a custom IP address here, but you must also add the IP address to a host interface. Customizing this parameter preempts MicroShift from adding a default IP address to the **br-ex** network interface.



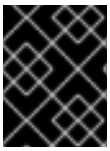
IMPORTANT

If you customize the **advertiseAddress** IP address, make sure it is reachable by the cluster when MicroShift starts by adding the IP address to a host interface.

If unset, the default value is set to the next immediate subnet after the service network. For example, when the service network is **10.43.0.0/16**, the **advertiseAddress** is set to **10.44.0.0/32**.

1.2.3. Extending the port range for NodePort services

The **serviceNodePortRange** setting extends the port range available to NodePort services. This option is useful when specific standard ports under the **30000-32767** range need to be exposed. For example, if your device needs to expose the **1883/tcp** MQ Telemetry Transport (MQTT) port on the network because client devices cannot use a different port.



IMPORTANT

NodePorts can overlap with system ports, causing a malfunction of the system or MicroShift.

Consider the following when configuring the NodePort service ranges:

- Do not create any NodePort service without an explicit **nodePort** selection. When an explicit **nodePort** is not specified, the port is assigned randomly by the **kube-apiserver** and cannot be predicted.
- Do not create any NodePort service for any system service port, MicroShift port, or other services you expose on your device **HostNetwork**.
- Table one specifies ports to avoid when extending the port range:

Table 1.1. Ports to avoid.

Port	Description
22/tcp	SSH port
80/tcp	OpenShift Router HTTP endpoint
443/tcp	OpenShift Router HTTPS endpoint
1936/tcp	Metrics service for the openshift-router, not exposed today
2379/tcp	etcd port
2380/tcp	etcd port
6443	kubernetes API
8445/tcp	openshift-route-controller-manager
9537/tcp	cri-o metrics
10250/tcp	kubelet
10248/tcp	kubelet healthz port
10259/tcp	kube scheduler

1.3. ADDITIONAL RESOURCES

- Checking Greenboot status

CHAPTER 2. CLUSTER ACCESS WITH KUBECONFIG

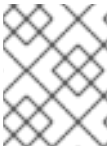
Learn about how **kubeconfig** files are used with MicroShift deployments. CLI tools use **kubeconfig** files to communicate with the API server of a cluster. These files provide cluster details, IP addresses, and other information needed for authentication.

2.1. KUBECONFIG FILES FOR CONFIGURING CLUSTER ACCESS

The two categories of **kubeconfig** files used in MicroShift are local access and remote access. Every time MicroShift starts, a set of **kubeconfig** files for local and remote access to the API server are generated. These files are generated in the **/var/lib/microshift/resources/kubeadmin/** directory using preexisting configuration information.

Each access type requires a different authentication certificate signed by different Certificate Authorities (CAs). The generation of multiple **kubeconfig** files accommodates this need.

You can use the appropriate **kubeconfig** file for the access type needed in each case to provide authentication details. The contents of MicroShift **kubeconfig** files are determined by either default built-in values or a **config.yaml** file.



NOTE

A **kubeconfig** file must exist for the cluster to be accessible. The values are applied from built-in default values or a **config.yaml**, if one was created.

Example contents of the kubeconfig files

```
/var/lib/microshift/resources/kubeadmin/
├── kubeconfig (1)
├── alt-name-1 (2)
│   └── kubeconfig
├── 1.2.3.4 (3)
│   └── kubeconfig
└── microshift-rhel9 (4)
    └── kubeconfig
```

1. Local host name. The main IP address of the host is always the default.
2. Subject Alternative Names for API server certificates.
3. DNS name.
4. MicroShift host name.

2.2. LOCAL ACCESS KUBECONFIG FILE

The local access **kubeconfig** file is written to **/var/lib/microshift/resources/kubeadmin/kubeconfig**. This **kubeconfig** file provides access to the API server using **localhost**. Choose this file when you are connecting the cluster locally.

Example contents of kubeconfig for local access

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://localhost:6443
```

The **localhost kubeconfig** file can only be used from a client connecting to the API server from the same host. The certificates in the file do not work for remote connections.

2.2.1. Accessing the MicroShift cluster locally

Use the following procedure to access the MicroShift cluster locally by using a **kubeconfig** file.

Prerequisites

- You have installed the **oc** binary.

Procedure

1. Optional: to create a **~/.kube/** folder if your RHEL machine does not have one, run the following command:

```
$ mkdir -p ~/.kube/
```

2. Copy the generated local access **kubeconfig** file to the **~/.kube/** directory by running the following command:

```
$ sudo cat /var/lib/microshift/resources/kubeadmin/kubeconfig > ~/.kube/config
```

3. Update the permissions on your **~/.kube/config** file by running the following command:

```
$ chmod go-r ~/.kube/config
```

Verification

- Verify that MicroShift is running by entering the following command:

```
$ oc get all -A
```

2.3. REMOTE ACCESS KUBECONFIG FILES

When a MicroShift cluster connects to the API server from an external source, a certificate with all of the alternative names in the SAN field is used for validation. MicroShift generates a default **kubeconfig** for external access using the **hostname** value. The defaults are set in the **<node.hostnameOverride>**, **<node.nodeIP>** and **api.<dns.baseDomain>** parameter values of the default **kubeconfig** file.

The **/var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig** file uses the **hostname** of the machine, or **node.hostnameOverride** if that option is set, to reach the API server. The CA of the **kubeconfig** file is able to validate certificates when accessed externally.

Example contents of a default kubeconfig file for remote access

```
clusters:
```

```
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://microshift-rhel9:6443
```

2.3.1. Remote access customization

Multiple remote access **kubeconfig** file values can be generated for accessing the cluster with different IP addresses or host names. An additional **kubeconfig** file generates for each entry in the **apiServer.subjectAltNames** parameter. You can copy remote access **kubeconfig** files from the host during times of IP connectivity and then use them to access the API server from other workstations.

2.4. GENERATING ADDITIONAL KUBECONFIG FILES FOR REMOTE ACCESS

You can generate additional **kubeconfig** files to use if you need more host names or IP addresses than the default remote access file provides.

IMPORTANT

You must restart MicroShift for configuration changes to be implemented.

Prerequisites

- You have created a **config.yaml** for MicroShift.

Procedure

1. Optional: You can show the contents of the **config.yaml**. Run the following command:

```
$ cat /etc/microshift/config.yaml
```

2. Optional: You can show the contents of the remote-access **kubeconfig** file. Run the following command:

```
$ cat /var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig
```

IMPORTANT

Additional remote access **kubeconfig** files must include one of the server names listed in the Red Hat build of MicroShift **config.yaml** file. Additional **kubeconfig** files must also use the same CA for validation.

3. To generate additional **kubeconfig** files for additional DNS names SANs or external IP addresses, add the entries you need to the **apiServer.subjectAltNames** field. In the following example, the DNS name used is **alt-name-1** and the IP address is **1.2.3.4**.

Example config.yaml with additional authentication values

```
dns:
  baseDomain: example.com
node:
  hostnameOverride: "microshift-rhel9" (1)
```

```
  nodeIP: 10.0.0.1
apiServer:
  subjectAltNames:
  - alt-name-1 2
  - 1.2.3.4 3
```

1 Hostname

2 DNS name

3 IP address or range

4. Restart MicroShift to apply configuration changes and auto-generate the **kubeconfig** files you need by running the following command:

```
$ sudo systemctl restart microshift
```

5. To check the contents of additional remote-access **kubeconfig** files, insert the name or IP address as listed in the **config.yaml** into the **cat** command. For example, **alt-name-1** is used in the following example command:

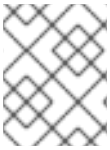
```
$ cat /var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig
```

6. Choose the **kubeconfig** file to use that contains the SAN or IP address you want to use to connect your cluster. In this example, the **kubeconfig** containing `alt-name-1` in the **cluster.server** field is the correct file.

Example contents of an additional kubeconfig file

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://alt-name-1:6443 1
```

1 The **/var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig** file values are from the **apiServer.subjectAltNames** configuration values.



NOTE

All of these parameters are included as common names (CN) and subject alternative names (SAN) in the external serving certificates for the API server.

2.4.1. Opening the firewall for remote access to the MicroShift cluster

Use the following procedure to open the firewall so that a remote user can access the MicroShift cluster. This procedure must be completed before a workstation user can access the cluster remotely.

For this procedure, **user@microshift** is the user on the MicroShift host machine and is responsible for setting up that machine so that it can be accessed by a remote user on a separate workstation.

Prerequisites

- You have installed the **oc** binary.
- Your account has cluster administration privileges.

Procedure

- As **user@microshift** on the MicroShift host, open the firewall port for the Kubernetes API server (**6443/tcp**) by running the following command:

```
[user@microshift]$ sudo firewall-cmd --permanent --zone=public --add-port=6443/tcp &&
sudo firewall-cmd --reload
```

Verification

- As **user@microshift**, verify that MicroShift is running by entering the following command:

```
[user@microshift]$ oc get all -A
```

2.4.2. Accessing the MicroShift cluster remotely

Use the following procedure to access the MicroShift cluster from a remote workstation by using a **kubeconfig** file.

The **user@workstation** login is used to access the host machine remotely. The **<user>** value in the procedure is the name of the user that **user@workstation** logs in with to the MicroShift host.

Prerequisites

- You have installed the **oc** binary.
- The **user@microshift** has opened the firewall from the local host.

Procedure

1. As **user@workstation**, create a **~/.kube/** folder if your RHEL machine does not have one by running the following command:

```
[user@workstation]$ mkdir -p ~/.kube/
```

2. As **user@workstation**, set a variable for the hostname of your MicroShift host by running the following command:

```
[user@workstation]$ MICROSHIFT_MACHINE=<name or IP address of MicroShift machine>
```

3. As **user@workstation**, copy the generated **kubeconfig** file that contains the host name or IP address you want to connect with from the RHEL machine running MicroShift to your local machine by running the following command:

```
[user@workstation]$ ssh <user>@$MICROSHIFT_MACHINE "sudo cat
/var/lib/microshift/resources/kubeadmin/$MICROSHIFT_MACHINE/kubeconfig" >
~/.kube/config
```

NOTE

To generate **kubeconfig** files for this step, see the "Generating additional kubeconfig files for remote access" link in the additional resources section.

1. As **user@workstation**, update the permissions on your **~/.kube/config** file by running the following command:

```
$ chmod go-r ~/.kube/config
```

Verification

- As **user@workstation**, verify that MicroShift is running by entering the following command:

```
[user@workstation]$ oc get all -A
```

CHAPTER 3. CHECKING GREENBOOT SCRIPTS STATUS

To deploy applications or make other changes through the MicroShift API with tools other than **kustomize** manifests, you must wait until the Greenboot health checks have finished. This ensures that your changes are not lost if Greenboot rolls your **rpm-ostree** system back to an earlier state.

The **greenboot-healthcheck** service runs one time and then exits. After Greenboot has exited and the system is in a healthy state, you can proceed with configuration changes and deployments.

3.1. CHECKING THE STATUS OF GREENBOOT HEALTH CHECKS

Check the status of Greenboot health checks before making changes to the system or during troubleshooting. You can use any of the following commands to help you ensure that Greenboot scripts have finished running.

Procedure

- To see a report of health check status, use the following command:

```
$ systemctl show --property=SubState --value greenboot-healthcheck.service
```

  - An output of **start** means that Greenboot checks are still running.
  - An output of **exited** means that checks have passed and Greenboot has exited. Greenboot runs the scripts in the **green.d** directory when the system is a healthy state.
  - An output of **failed** means that checks have not passed. Greenboot runs the scripts in **red.d** directory when the system is in this state and might restart the system.

- To see a report showing the numerical exit code of the service where **0** means success and non-zero values mean a failure occurred, use the following command:

```
$ systemctl show --property=ExecMainStatus --value greenboot-healthcheck.service
```

- To see a report showing a message about boot status, such as **Boot Status is GREEN - Health Check SUCCESS**, use the following command:

```
$ cat /run/motd.d/boot-status
```